Illumination

Sandro Spina
Computer Graphics and Simulation Group

Computer Science Department
University of Malta

Illumination … *for 3D real-time rendering*

- During this module we'll discuss the basics of *illumination* theory.
- We'll go through the computations required to enable real-time rendering of 3D scenes.
- Rendering Authenticity can be split in 2:
  - Geometry
  - Visual Appearance – i.e. How does light interact with the surface of the geometry.
- The closer we model light interaction with surfaces the higher the perception of reality we are able to produce (photorealism).
- However since we are talking about real-time performance we'll have to make some tradeoffs.
- Module based on Akenine-Mueller's Real-Time Rendering book

Illumination ... models

- In practice we need to come up with two model categories
  - A surface model
  - A lighting model

- We'll see that there are many surface models ...
- And even mode lighting models ...

- Local Illumination versus Global Illumination
- In this module we'll be looking at local illumination models however ...
- Towards the end of the course you'll also be given a 3hr introduction to Global Illumination and Physically Based Modelling.

Visual (Physical) Phenomena

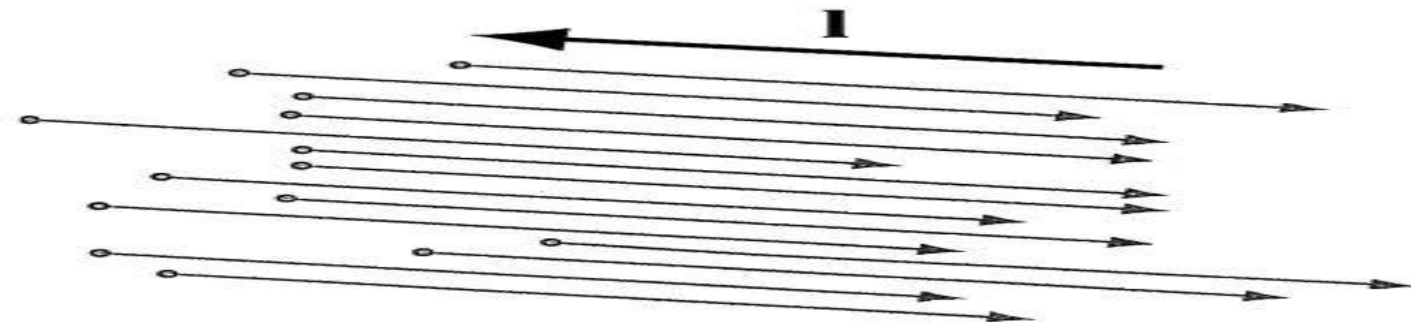
- If our goal is to model reality (in terms of lighting and surfaces) then we need to understand some physical properties (phenomena) of light.

- **Light is emitted, scattered and absorbed –**
  - Light is *emitted* by the sun or other sources (flames, energy saving bulb, etc)
  - Light interacts with objects; part is absorbed, part is *scattered* and propagates in new directions,
  - Light is *absorbed* by a sensor (human eye, film, CCD in digital cameras)

- Just look around to observe all these phenomena

Light Sources (Directional)

- Light sources generate *electromagnetic radiant energy* that travels through space.
- Different light sources exists however we can group these sources in different clusters.
- To start, let’s create a model for light emitted by extremely distant sources (like the sun) as **directional lights**.
- Directional lights have two basic properties – a direction (to light) vector. Referred to as the *light vector*. And intensity of light.



Light Sources (Intensity of ...)

- Every light source has an associate amount of illumination that it emits.

- The science of measuring light is called radiometry (you'll have a taste of this when doing PBR). Light is not colour but waves ... Colour is the interpretation given by our visual system to the visible range of frequencies.

- We can approximate the value of the intensity by measuring power through a unit area surface (plane) perpendicular to the light vector l. This quantity is called **irradiance**.

- Ultimately we view light as colour .. Hence we can directly colour light and represent irradiance as an RGB value containing three number representing red, green and blue.

Light Sources (Ambient)

- Even if you have just one light source in your room (or whatever) light emitted from that light source is dispersed and some of it bounces off the surfaces of other objects ...
- We shall not calculate this value (for now) but we shall take an approximation to this value by specifying an ambient light.
- Ambient light essentially covers surrounding and environmental lighting ... Without the complex computations.
- IMPORTANT: it is not a physically based value. But an approximation.

Light hitting a surface ...

- Ok, now we know what irradiance is; which gives us a measure of the amount of light leaving a particular light source (how bright the source is ...)

- It is a measure of how much light is travelling in space ...

- We now need to compute the amount of light which is hitting a surface and at a particular point on the surface.
- This is know as the **surface irradiance**.

- SI is equal to:
  - The radiance measured perpendicular to the light vector *l*, multiplied by the cosine of the angle between *l* and the surface normal vector *n.*

Light hitting a surface … (diagram)

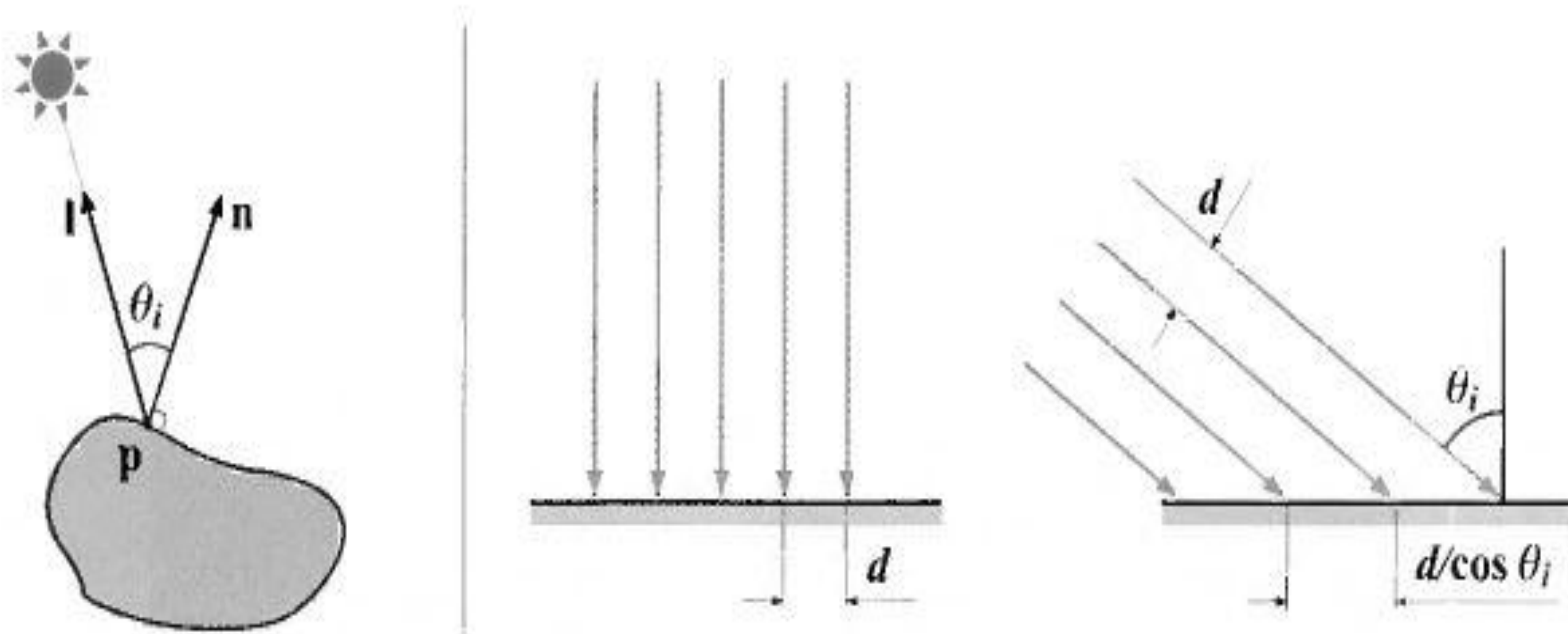


Figure 5.4. The diagram on the left shows the lighting geometry. In the center, light is shown hitting a surface straight-on, and on the left it is shown hitting the surface at an angle.

Geometric interpretation of the cosine factor

- This is a very important concept in the realistic simulation of light hitting a surface ...
- Irradiance (at a point p on a surface) is:
  - Proportional to the density of the rays and
  - Inversely proportional to the distance *d* between them.
- Note that the distance between the rays on the surface is given by: $\mathbf{d / \cos \theta}$
- Check this out with a number of examples
- The bigger the angle the dimmer the brightness ...

Computing irradiance on a surface … (from one source)

- With the knowledge acquired in the previous slides we can calculate how much irradiance is falling on every surface point p of our 3D objects.
- Irradiance $E = E_l \cos \theta$
  - where E_l is equal to the irradiance perpendicular to the light vector l
  - Note that $\cos \theta$ needs to be clamped to non-negative values

- Recall the dot product and its use to determine the angle between two vectors.
- The cosine between **l** and **n** can be computed by taking their dot product.
- **Irradiance E =** $E_l \max(n.l, 0)$

Computing irradiance on a surface ... (from multiple sources)

- Irradiance is additive ... This means that the total irradiance from two or more sources is simple the sum of the irradiance quantities from each light source.
- This translates to:

$$\textbf{Irradiance E} = \sum_{k=1}^{n} El_k \, cos\theta$$

- Assuming above that we have n directional lights.

Materials (or how objects appear)

- We've seen now how to compute the amount of irradiance falling on a surface ....
- However different surfaces have different properties and we need to model these as well in order to get visually pleasing and realistic results.
- Object appearance is portrayed by attaching materials to models in the scene.
- Each material is associated with a set of shaders, textures and other properties which describe how a surface is rendered.
- These are used to simulate light interaction with the object surface.
- In this module we'll investigate a simple material model.

Scattering and Absorption

- All light-matter interactions are a result of two phenomena
  - **Scattering**
  - Causes light to change direction.
  - Happens when light encounters some form of optical discontinuity.
  - Light scattering can either be
    - Reflection (out of the surface)
    - Refraction (into the surface)

  - **Absorption**
  - Happens inside matter and causes some of the light to be converted into another kind of energy and disappear.
  - Reduces the amount of light
  - Does not change the direction of light

Scattering and Absorption … (diagram)

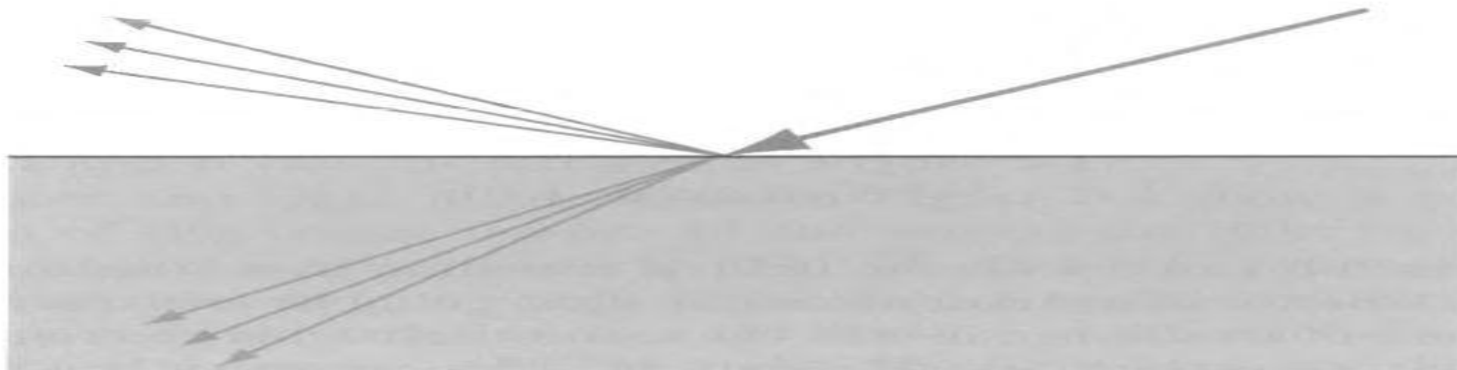


Figure 5.6. Light scattering at a surface—reflection and refraction.

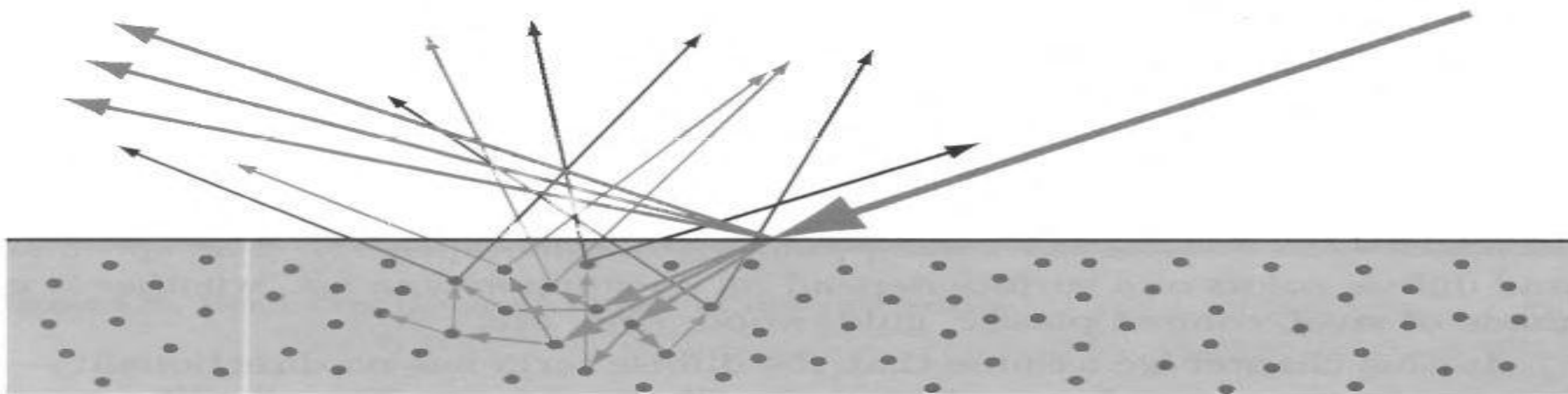


Figure 5.7. Interactions with reflected and transmitted light.

Specular and Diffuse

- As discussed in the previous slide the light that is reflected immediately at the surface has a different direction distribution and colour to the light that was first partially absorbed and then scattered back out.

- We can thus calculate these two quantities separately then add them together.

- We refer to these two components as:
  - **The Diffuse Component (term) –** representing light that has undergone transmission, absorption and scattering.
  - **The Specular Component (term) –** representing light that was reflected at the surface

Outgoing Light ...

- Irradiance refers to the incoming illumination ... Which we use to calculate the outgoing light.
- We refer to outgoing light as **exitance (M)** which is measured with the same units of irradiance (energy/second/unit area)
- Light-matter interactions are *linear*
  - *Meaning that if double the amount of irradiance falls on a point, existance will double as well for a particular material.*
  - *Leading us to define one of the properties of materials as ...*
- *A Ratio between Exitance and Irradiance*
- *An important characteristic of every material hence is the value of the exitance divided by the irradiance.*

Exitance/Irradiance Ratio

- This ratio can vary for light of different colours, so it is represented as an RGB vector or colour, commonly called the *surface colour* ***c*****.**
- Surface colour is composed (in shading equations) of the two terms we just saw namely the diffuse colour component c_{diff} and specular colour component c_{spec}.
- These terms determine important material properties. The specular and diffuse colours of a surface depend on its composition.
  - *Eg. Steel, colored plastic, gold, wood, skin, etc …*

Directionality of Reflections

- For the diffuse component we'll be assuming that there is no directionality ... i.e. Light is scattered back equally in all directions. NOTE that this is an approximation and is not physically-based.
- On the contrary the specular terms does have significant directionality (specular highlights) that need to be addressed.
- Unlike colour ... Specular reflections depend on the smoothness of the surface.
- A smoother surface exhibits sharper reflections and narrow highlights (mirror) whereas a not so smooth and rougher surface (brushed metal, CDR, etc) shows blurry reflections and relatively broad dim highlights.

Lambert's Law and the Dot Product

- The dot product is heavily used in lighting calculations. Recall that
  - r.s = ||r|| . ||s|| cos (b)

- Of significant importance is Lambert's Law which states that the intensity of illumination on a diffuse surface is proportional to the cosine of the angle between the surface normal vector and the light source direction.

- For eg. given a source of light situated at (20, 20, 40) and the illuminated point is (0, 10, 0) with a normal vector of $[0, 1, 0]^T$.

- The direction of the light source respect to the surface point is defined by the vector s: [20-0, 20-10, 40-0] = [20, 10, 40]

- Magnitude of s = || s || = $\sqrt{20^2 + 10^2 + 40^2}$ = 45.826

- Therefore we have 1 X 45.826 X cos(b) = 0x20 +1x10 + 0x40
- Cos (b) = 10 / 45.826 = 0.218 .... This means that the light intensity at (0, 10, 0) is 0.28 the intensity at (20,20,40)

Smoothness of material

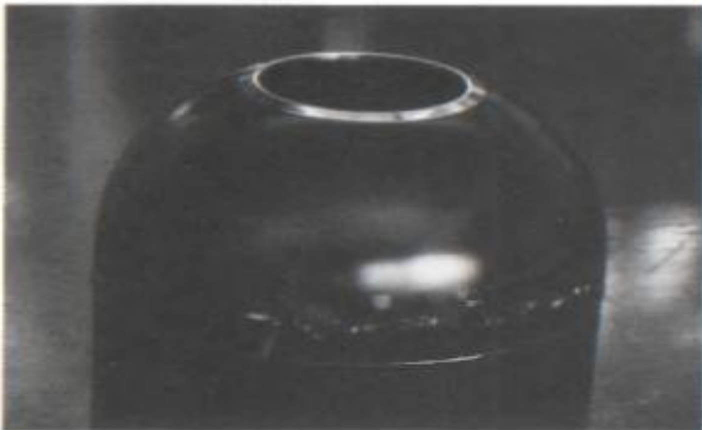


Figure 5.8. Light reflecting off surfaces of different smoothness.

Types of Light Sources

- In a virtual scene many different light sources are possible. The most common types include
  - Directional Light
  - Spot Light
  - Point Light
- A directional light is one that comes from one particular direction and whose distance is considered infinite.
- In nature such light does not exist but can for eg. simulate the light (from the sun) coming in through a window.

Types of Light Sources (ii)

- Point lights are similar to a directional lights, however a point light fades as the distance between the source of the light and the point being shaded (on the geometry) increases.

- This property is referred to as <u>distance attenuation</u>.

- Point Lights have a position in space, a distance attenuation factor, and they shine light in <u>all directions</u>.

- Examples of points lights include a candle, a bulb, etc …

Types of Light Sources (iii)

- The third type of light simulates spot lights.
- A spot light is similar to a point light, with the difference being that a spot light only shines in a specific direction within a cone volume.
- The most common examples of spot lights used in games are probably flashlights and car headlamps.

Shading

- *Shading* is the process of using an equation to compute the outgoing radiance L_o (NOT exitance M) along the view ray, **v**, based on material properties and light sources.

- We have already discussed material properties, light sources and exitance. We shall now see how everything is put in together and implemented.

- In CG there are usually three levels of computing lighting:
  - Per Primitive (Flat Shading)
  - Per Vertex (Gouraud Shading)
  - Per Pixel (Phong Shading)

Flat Shading

- When rendering using flat shading lighting calculations are carried out once for the whole primitive.
- Only one normal is necessary …
- Discontinuities in shading are very apparent and does not produce a perceptually realistic render.
- Hardly used any more with today’s hardware performance.
- Also referred to as per-primitive shading.

Per-Vertex Shading (Gouraud)

- In per-vertex shading, lighting calculations are carried out for each vertex in a mesh.
- Usually the results of the per-vertex light calculations are either combined with the vertex colour, or act the colour, and their values are interpolated across the surface of the primitive.
- The positive feature is thus that the calculations are limited to the total number of vertices present sent down the pipeline. Using standard lighting algorithms this is fairly cheap on today's hardware.
- The downside however is rendering quality … which is tied directly to the vertex count, which requires a large number of vertices to obtain best quality. Adding more geometry however is not ideal for real-time performance !!

Per- Pixel Shading (Phong)

- Per-pixel lighting involves lighting calculations performed on each raster pixel that makes up the surface.
- Since there are (usually) more pixels that make up a surface than vertex points, this will always give a tremendous amount of quality regardless of how many polygons and vertices are present in the scene.
- As you can imagine then … screen resolution plays an important role.
- We shall be using per-pixel shading in the examples we'll be looking at today.
- Per-pixel shading is carried out in the pixel shader … whereas vertex shaders are used to calculate the colours of vertices in per-vertex shading.

Flat vs Gouruad vs Phong

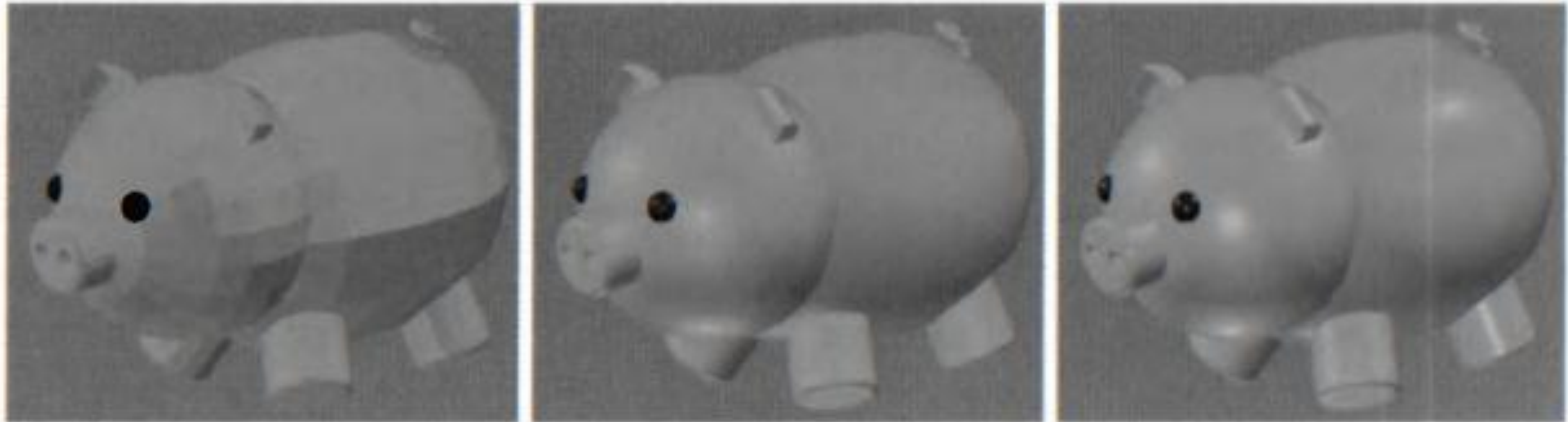


Figure 5.17. Flat, Gouraud, and Phong shading. The flat-shaded image has no specular term, as shininess looks like distracting flashes for flat surfaces as the viewer moves. Gouraud shading misses highlights on the body and legs from this angle because it evaluates the shading equation only at the vertices.

Light Models ...

- Lighting models (used by shading models) are algorithms used to perform lighting calculations.
- There exist many different types of lighting models addressing different components of the lighting … we shall be looking at three of them.
- Lambert Diffuse Model: Using as we’ve seen in a preceding slide Lambert’s law to calculate the diffuse component of lighting.
- Phong Specular Model: This model is used to calculate specular highlights on shiny objects.
- Blinn-Phong Specular Model: This model is a variation of Phong which provides a cheaper implementation.

Light Terms … (i)

- Local Illumination: As opposed to global illumination which takes into account all the lights in a scene and how light is bouncing off on all the surface throughout the scene to calculate light falling at a surface point, local illumination does not take into account the light that is bouncing off all the surface.

- Emissive: used to represent how much light is given off by a surface .. Usually denoted by the constant *e*

- Ambient: Light component simulating global illumination … this is usually a constant colour used for all the objects in the scene

- Diffuse: refers to light that has reflected and scattered after hitting a surface in many different directions. The viewing angle does not effect how the surface looks (obviously the normal and light direction do)

Light Terms … (ii)

- Specular: refers to light which is similar to diffuse light, but with the exception that the lights is now reflected in the (or near the) mirror direction of the incoming light.

- Attenuation: used to represent how light loses intensity over distance. This value is used in light sources such as point and spot lights. Value is usually between 0 and 1.

- Combining all these lighting terms we get a simple equation used to try to approximate light as seen in real life …

- Light = Emissive + Ambient * (Diffuse + Specular) * Attenuation, where Diffuse = kd * diffuse_calculation and Specular = ks * specular_calculation. Ks and Kd are determed from the surface material.

Diffuse Lambert Shader

- We shall now have a look at a (pixel) shader which we'll use to compute diffuse component of lighting.
- We use Lambert's Law to calculate the diffuse component, there we need to know two vector for the calculation, namely, the normal and light vectors.
- Recall that if the dot product between these two vectors is 1 than we have two perpendicular vectors pointing in the same direction.
- The light would be fully illuminating this point.
- Let us define angle = dot(normal, light_dir), then:
- Diffuse = light_color * angle

OpenGL Vertex Shader

- Varying vec3 normal;
- Varying vec3 lightVec;

- Uniform vec3 lightPos;

- Void main()
- {

  Gl_position = glModelViewProjectionMatrix * gl_Vertex;

  Vec4 pos = gl_ModelViewMatrix * gl_Vertex;

  Normal = gl_NormalMatrix * gl_Normal;

  lightVec = lightPos – pos;
- }

OpenGL Pixel Shader

- Varying vec3 normal;
- Varying vec3 lightVec;

- Void main()
- {
- Normal = normalize(normal);
- lightVec = normalize(lightVec);

- Float diffuse = saturate(dot(normal, lightVec));

- Gl_FragColor = vec4(1, 1, 1, 1) * diffuse;

DirectX HLSL Vertex/Pixel Shader

```
float4x4 worldView : WorldView;
float4x4 worldViewProj : WorldViewProjection;
float4 lightPos;

struct Vs_Input
{
   float3 vertexPos  : POSITION;
   float3 norm       : NORMAL;
   float2 tex0       : TEXCOORD0;
};

struct Vs_Output
{
   float4 vertexPos  : POSITION;
   float3 norm       : TEXCOORD0;
   float3 lightVec   : TEXCOORD1;
};
```

DirectX HLSL Vertex/Pixel Shader

```
struct Ps_Output
{
    float4 color     : COLOR;
};

Vs_Output VertexShaderEffect(Vs_Input IN)
{
    Vs_Output vs_out;
    float4 Pos = mul(worldView, float4(IN.vertexPos, 1));
    vs_out.vertexPos = mul(worldViewProj, float4(IN.vertexPos, 1));
    vs_out.norm = mul(worldView, IN.norm);
    vs_out.lightVec = lightPos.xyz - Pos.xyz;
    return vs_out;
}
```

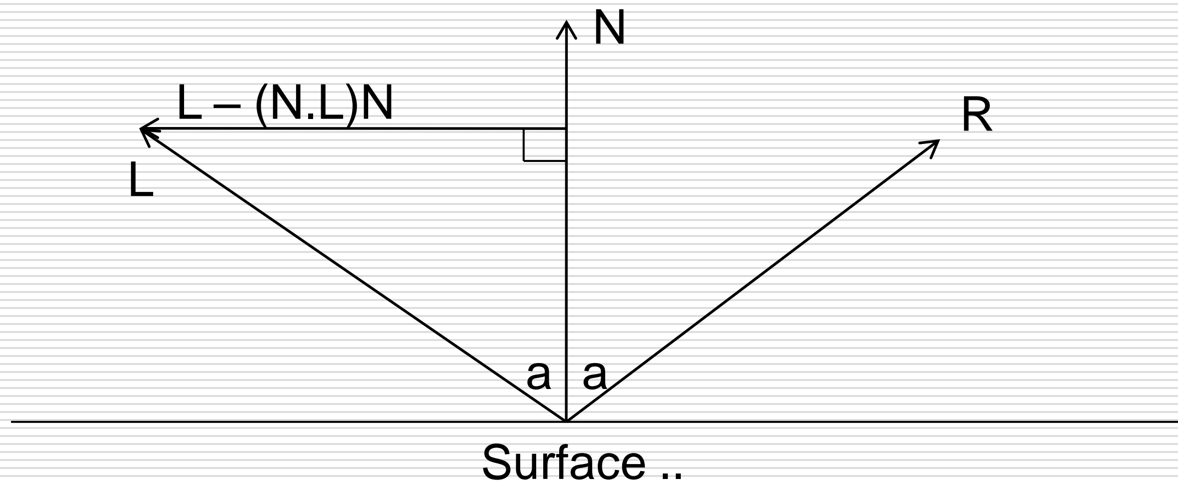
DirectX HLSL Vertex/Pixel Shader

```
Ps_Output PixelShaderEffect(Vs_Output IN)
{
    Ps_Output ps_out;
    float3 normal = normalize(IN.norm);
    float3 lightVec = normalize(IN.lightVec);
    float diffuse = saturate(dot(normal, lightVec));
    ps_out.color = float4(1, 1, 1, 1) * diffuse;
    return ps_out;
}

technique LambertDiffuse
{
  pass Pass0
  {
    VertexShader = compile vs_2_0 VertexShaderEffect();
    PixelShader  = compile ps_2_0 PixelShaderEffect();
  }
}
```

Reflection Vector

- A specular reflection is dependant on the view as well therefore the view vector needs to be factored into the lighting equation for the specular term.
- We also need to calculate the reflection vector of L (the light vector) which gives the light vector reflected off the surface normal.
- $Perp_N L = L - (N.L)N$ therefore $R = L - 2\, perp_N L = 2\,(N.L)N - L$



Phong Lighting

- As the view and the reflected light vector become more perpendicular, the highlight becomes brighter. You can check this our on the whiteboard!!
- As always we use the dot product to determine the angle between these two vectors.
- This dot product is usually raised to a power to simulate surface roughness.
- Specular = power(dot(V, R), 8)
- Note that if the diffuse value is 0 then we don't need to calculate the specular component.
- Final color = diffuse component + specular component

Phong Lighting Shader – HLSL

```
float4x4 worldView : WorldView;
float4x4 worldViewProj : WorldViewProjection;
float4 lightPos;
float4 eyePos;

struct Vs_Input
{
   float3 vertexPos  : POSITION;
   float3 norm       : NORMAL;
   float2 tex0       : TEXCOORD0;
};
struct Vs_Output
{
   float4 vertexPos  : POSITION;
   float3 norm       : TEXCOORD0;
   float3 lightVec   : TEXCOORD1;
   float3 viewVec    : TEXCOORD2;
};
```

Phong Lighting Shader – HLSL

```
struct Ps_Output
{
    float4 color      : COLOR;
};

Vs_Output VertexShaderEffect(Vs_Input IN)
{
    Vs_Output vs_out;
    float4 Pos = mul(worldView, float4(IN.vertexPos, 1));
    vs_out.vertexPos = mul(worldViewProj, float4(IN.vertexPos, 1));
    vs_out.norm = mul(worldView, IN.norm);
    vs_out.lightVec = lightPos.xyz - Pos.xyz;
    vs_out.viewVec = eyePos.xyz - Pos.xyz;
    return vs_out;
}
```

Phong Lighting Shader – HLSL

```
Ps_Output PixelShaderEffect(Vs_Output IN)
{
   Ps_Output ps_out;
   float3 normal = normalize(IN.norm);
   float3 lightVec = normalize(IN.lightVec);
   float3 viewVec = normalize(IN.viewVec);
   float diffuse = saturate(dot(normal, lightVec));

   float3 r = normalize(2 * diffuse * normal - lightVec);
   float specular = pow(saturate(dot(r, viewVec)), 8);

   float4 white = float4(1, 1, 1, 1);
   ps_out.color = white * diffuse + white * specular;
   return ps_out;
}
```

Blinn – Phong Lighting

- The Blinn-Phong lighting model proposed by Jim Blinn, removes the expensive (not that much on today's hardware) vector reflection calculation.
- Instead, in this lighting model (for specular highlights) Blinn uses the half-vector between the light and view directions.
- This is then used in the dot product instead of the reflection vector.
- Half Vector = normalise(lightVec + viewVec)
- Specular = pow(saturate(dot(normal, halfVector)), 50);
- Not as good (and perceptually accurate) as Phong but much less expensive to compute. Differences as still hard to notice (for eg during a game)

Point Light Shader (properties)

- Float4x4 World;
- Float 4x4 View;
- Float 4x4 Projection;
- Float3 AmbientLightColor = float3(.15, .15, .15);
- Float3 DiffuseColor = float3(.85, .85, .85);
- Float3 LightPosition = float3(0, 0, 0);
- Float LightColor = float3(1, 1, 1);
- Float LightAttenuation = 5000;
- Float LightFalloff = 2;

- Texture BasicTexture;
- Sampler BasixTextureSampler = sampler_state {
  texture = <BasicTexture>;
- };

Point Light Shader (structs)

```
Struct VertexShaderInput
{
    float4 Position:POSITION0;
    float2 UV:TEXCOORD0;
    float3 Normal:NORMAL0;
};

Struct VertexShaderOutput
{
    float4 Position:POSITION0;
    float UV:TEXCOORD0;
    float3 Normal:TEXCOORD1;
    float4 WorldPosition:TEXCOORD2;
};
```

Point Light (Vertex Shader)

```
VertexShaderOutput VertexShaderFunction(VertexShaderInput input)
{
    VertexShaderOutput output;
    Float4 worldPosition = mul(input.Position, World);
    Float4 viewPosition = mul(worldPosition, View);
    Output.Position = mul(viewPosition, Projection);

    Output.WorldPosition = worldPosition;
    Output.UV = input.UV;
    Output.Normal = mul(input.Normal, World);

    Return output;
}
```

Point Light (Pixel Shader)

```
Float4 PixelShaderFunction(VertexShaderOutput output) : Color0
{
    Float3 diffuseColor = DiffuseColor;
    If (TextureEnabled) diffuseColor *= tex2D(BasicTextureSampler, input.UV).rgb;
    Float3 totalLight = float3(0, 0, 0);
    Float3 totalLight += AmbientLightColor
    Float lightDir = nomralize(LightPosition – input.WorldPosition);
    Float diffuse = saturate(dot(normalize(input.Normal), lightDir);
    Float d = distance(LightPosition, input.WorldPosition);
    Float att = 1 – pow(clamp(d / LightAttenuation, 0, 1), lightFallOff);

    totalLight += diffuse * att * LightColor;

    Return float4(diffuseColor * totalLight, 1);
}
```